Initiation à Python pour la théorie des graphes

1 Prise en main de Python

1.1 Le langage Python

Python est un langage de programmation et de scripting. Ca veut dire qu'il peut être utilisé à la place de langages comme C, C++ ou Java pour des développements assez conséquents; mais c'est aussi un remplacant de Perl ou même de bash, dans son rôle de "couteau suisse du programmeur". Sa souplesse explique qu'il a beaucoup utilisé pour le Web côté serveur; mais ca n'est de loin pas son seul domaine d'application. Par exemple, on trouve beaucoup de code mathématique réalisé en Python. Python a aussi pris une place non-négligeable dans le segment du code embarqué. C'est un bon nouveau couteau suisse!
Les particularités de Python que vous devez connaître pour vous lancer sont les suivantes:

- C'est un langage dynamiquement typé. Cela signifie que les types des données sont attachés aux valeurs manipulées, et non pas aux variables, comme c'est le cas pour les langages statiquement typés. Vous pouvez donc affecter un nombre à une variable qui précédemment contenait une chaîne de caractères.
- Python est muni d'une bonne bibliothèque standard; elle est complétée par de nombreux paquets logiciels qui peuvent être téléchargés sur Internet.
- C'est un langage orienté objet. Certains types de données de la bibliothèque standard sont définis via un mécanisme de classes, qui supporte une notion d'héritage.
- Python a un système de modules assez propre; en cela, il ressemble plus à Java qu'à C.
- Python est un langage interprété. Cela signifie que vous pouvez charger votre code source dans un interprète pour le tester. Il est bien sûr possible de lancer une exécution en ligne de commande. Aucune compilation ou manipulation de fichiers objets, et aucune édition de lien ne sont nécessaires pour cela.
- Parce qu'il est possible de travailler dans l'interprète, l'utilisation de print() est rarement indispensable. L'expérience montre qu'utiliser des fonctions d'affichage rend le code moins souple et moins extensible. Il vous est recommandé d'essayer de coder "sans print()", en particulier en utilisant le mécanisme de valeur de retour de fonctions.
- Python vous permet de retourner plusieurs valeurs de retours, en placant ces valeurs dans un couple, qui peut être affecté directement à deux vraibles au moment de récupérer ce couple.

1.2 Préparation

Pour prendre en main Python, vous aurez besoin:

- d'un interprète Python. Ca se trouve dans toutes les bonnes crèmeries! Le code fourni fonctionne avec les version V2 et V3 de Python. Soyez informés que les syntaxes de ces deux langages sont (légèrement) incompatibles. La version 3 est la plus récente et la plus mûre; mais beaucoup de bibliothèques de code écrites par des tiers n'existent que pour Python V2.
- de documentations. Vous trouverez des documents de référence pour le langage, la bibliothèque standard ainsi que des tutoriels sur le site officiel du langage Python.

1.3 Démarrage de l'interprète

Votre première difficulté est de démarrer l'interprète dans un contexte où le code fourni vous est accessible. Sous linux, il suffit de lancer Python dans le répertoire où se trouve ce code.
Si c'est bien le cas, la commande `from graphes import *` devrait être acceptée silencieusement par Python.
`2+2` vous répond une valeur, alors que `a = 2+2` ne répond rien. Mais juste `'a'` vous donne la valeur attendue.
En tapant `a, b = 2+2, 3*3`, vous voyez comment Python vous permet de manipuler des tuples directement. Egalement avec `x = 2, 3`: essayez ensuite `x[0]` pour récupérer les valeurs.
Les deux structures de données les plus utilisées dans Python sont les listes et les dictionnaires.

```
l = [1,2,3]
len(l)
l[0]
l.append(5)
l
m = l
n = list(l)
l.pop()
l
m
n
help(l)
type(l)
help(list)
dir(l)
d = {1:  'abc', 3:  'de'}
d[3]
d[2]
d[2] = 5
help(d)
dir(d)
```

2 Prise en main du code et réalisations

2.1 Fichier fourni

Il vous est proposé un unique fichier - graphes.py - contenant une unique fonction: graphe_oriente(). Cette fonction prend deux paramètres numériques. Le premier désigne le nombre de sommets du graphe à générer. Le second vous permet de générer différents graphes: variez-la pour varier les graphes obtenus. La génération est déterministe: elle ne fait pas intervenir le hasard.

2.2 Premières réalisations dans les graphes orientés

Après avoir tapé la commande d'import ci-dessus (`from graphes import *`), vous devez pouvoir générer un graphe G et le stocker dans une variable par l'appel suivant:
`g = graphe_oriente(100, 10)`

1. Sous quelle forme est encodé ce graphe? Combien a-t-il de sommets? Quels sont les successeurs du sommet 0? Quels sont les successeurs du sommet 67?

2. Ecrire une routine qui prend un graphe et un sommet du graphe, et retourne le degré sortant de ce sommet (i.e. le nombre de ses successeurs).

3. Ecrire une routine qui renvoie l'ensemble des sommets d'un graphe.

4. Ecrire une routine qui renvoie l'ensemble des arcs d'un graphe.

5. Ecrire une routine qui calcule le nombre de sommets et le nombre d'arcs d'un graphe, et renvoie ces deux nombres sous la forme d'un couple.

6. Ecrire une routine qui calcule le graphe miroir d'un graphe.

7. Ecrire une routine qui calcule le degré entrant d'un sommet (nombre de prédécesseurs).
On appelle *sous-graphe de* $\vec{G} = (V, A)$ *induit par* $S \subseteq V$ le graphe $\vec{H} = (S, B)$ où
$B = \{(x, y) \mid x \in S \text{ et } y \in S\}$. On le note $\vec{G}[S]$

8. Ecrire une routine qui calcule le sous-graphe de $\vec{G}$ induit par S.

2.3 Accessibilité

1. Coder la routine `Accessibles1()` vue en cours.

2. Donner un exemple de graphe et de sommet sur lesquels `Accessibles1()` fonctionne correctement; et un autre exemple montrant ses limites.

3. Coder la routine `Accessibles2()` vue en cours. Ecrire une fonction `Appel_accessibles2()` permettant de l'appeler selon le même schéma d'appel que `Accessibles1()`. Vérifier qu'elle fonctionne également sur votre contre-exemple trouvé précédemment.

4. Coder la routine `Accessibles3()`, ainsi qu'une fonction `Appel_accessibles3()` permettant de l'appeler de la même manière que `Accessibles1()`.

5. Implémentez la routine `Reconstruire_chemins()`, et illustrez son bon fonctionnement.

6. Coder une routine `Accessible(G, s, t)` prenant un graphe $\vec{G}$ et deux sommets s et t, et retournant un chemin de s à t lorsqu'il y en a un; et le symbole `None` lorsqu'il n'y en a pas.

7. Ecrire une routine `Verifier_chemin(G, s, t, C)`, prenant un graphe $\vec{G}$, deux sommets s et t, et un chemin C encodé comme une liste de sommets, et qui vérifie que C est bien un chemin de s à t dans $\vec{G}$. Illustrer son bon fonctionnement.
